Corporate Developer

BY CHRISTINE COMAFORD

A Guide to Controls And Window Types

Corporate developers designing a graphical user interface (GUI) often face a double-edge sword: A well-designed user interface leads to reduced end user training and support, while a poorly designed interface mires users in a mind-numbing plethora of haphazard controls and dialog boxes. Since there are a number of different types of controls and windows, the successful GUI designer needs to have a firm understanding of when it's appropriate to use one control over another. Just as the plenitude of cheap fonts has led to ransom-note typography, the number and quality of today's custom-development controls often lead developers to ransom-note control usage. In this column, you'll learn more about the basics of windows and controls that serve as the foundation for a well-designed user interface.

OPENING WINDOWS As the name of the operating environment would lead you to believe, windows are the fundamental objects through which most user interaction takes place. Like the wise man who built his house upon the rock, your Windows application needs a solid foundation upon which to add functionality.

As you develop Windows applications for other users within your company, your goal always should be to maintain as much consistency with other Windows applications as possible. Why force a user to learn some new or inconsistent action when you could make a design decision to use a standard control or behavior that's already known to the user? Let's start with a review of the kinds of windows that you can use in your application and when to use each one.

Window types include primary (also called *application* or *main*) windows, multiple document interface (MDI) windows, MDI child windows (also referred to as document windows), and dialog boxes. Applications routinely create and use other windows that include dialog boxes, MDI child windows, and non-MDI child windows.

CHOOSING THE RIGHT WINDOW If users will be working with a variable amount of data but just one type of data, use a standard main application window, as the Write and Notepad accessories do. If your program needs to provide access to multiple sets of data or to multiple views of one data set, make the main window an MDI window and display the data sets in MDI child windows. Program Manager is the most ubiquitous example of an MDI program. Finally, if your program's interaction with the user is simple and straightforward, consider using a simple dialog box as the main application window. The Calculator and Character Map accessories are two examples of programs whose main window is a dialog box. Figure 1 shows three of the four types of windows that are available to you:

In this column, you'll learn more about the basics of windows and controls that serve as the foundation for a well-designed user interface.

• **Application windows.** The main application window usually contains a resizable frame and a title bar. In addition, application windows should have a control-menu box, and minimize and maximize buttons. To make matters a bit more complicated, the main window can be just a dialog box, in which case the user won't be able to size the window frame. If your main application window is a dialog box, include a control-box menu and a minimize button but not a maximize button.

• **MDI windows.** An MDI window is capable of creating and managing multiple MDI child windows. Just as Windows manages a desktop full of main program windows, an MDI application manages a set of MDI child windows, also called document windows. Document windows can be tiled or cascaded within the MDI window, and they can be reduced to icons, again within the MDI window. The child windows always stay completely inside the main program window.

In general, if your application calls for complex data entry or the ability to view multiple data elements, use a main MDI window and create MDI child windows as necessary. Before you begin designing your application, spend some time thinking about the way you plan to present data to the user. If there's any chance that the user would need to open another window while halfway through the work process, choose the MDI window option. That way, your application can open up an MDI child window, let the user interact with the data, and then return to the original work.

• **MDI child windows.** The third kind of window is the MDI child window. MDI child windows share almost all the characteristics of an application window. They can be resized, minimized, and maximized but always within the main MDI window. When an MDI child window is maximized, its title is appended to the title of the main MDI window (for example, "Program Manager - [Main]").

The ability to open more than one MDI child window lets your application display more than one data set or more than one kind of data set. Users can either work on many data sets at once or

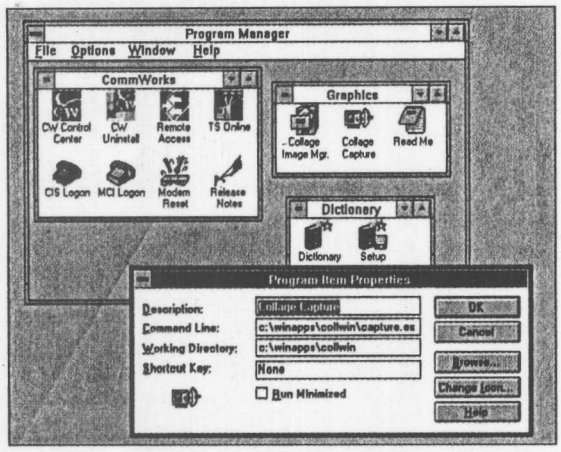

***Figure 1:* Three of the four available window types are shown in this figure. The main application is an MDI window that contains MDI child windows. The dialog box is also considered a window.**

easily compare the contents of similar forms. For example, consider a purchase-order application where, after working through the first half of the PO, the user might have to enter information about a customer. If the user tries to edit a customer record that doesn't exist, he shouldn't have to exit from the order screen and lose all of the data already entered just to enter a new customer. Why not just open an MDI child window to enter the new customer information, save the data, and then close the window before continuing?

Because the term *document* might be confusing, it's acceptable to call these windows by more appropriate names that refer to the context of the data. For example, Figure 2 shows Program Manager with two open document windows called Main and CommWorks. Although the Program Manager refers to these as group windows, they are still just MDI document windows. On the other hand, if the application is Microsoft Word for Windows or WordPerfect, it's correct to refer to these windows as documents.

As you design your applications, give some thought to how many MDI windows the user can open at one time. For example, if you're developing an application for entering purchase orders, you might permit the user to work on multiple purchase orders in multiple document windows at the same time. If you also let him have a customer window open for each purchase order, chances are good he'll be confused by all the windows splashed across the screen. I've found that providing multiple instances of the primary objects and single instances of the peripheral objects works well.

• **Dialog boxes.** Dialog boxes provide a two-way interaction between the application and the user. On one hand, applications use dialog boxes to get more information from the user before a command can be completed. The controls that you use in a dialog box collect information and pass that information back to the application. On the other hand, dialog boxes also present data to the user via message boxes and the initial values of controls displayed in the dialog box. Dialog boxes can have a minimize button (and perhaps they should if the dialog box is the application) and even a menu bar.

Let's take a moment to bone up on the three types of Windows dialog boxes: modeless, modal, and system modal. What you want your application to accomplish will dictate the kind of dialog box you implement. Choose modeless dialog boxes for on-going tasks or to present information that will be helpful to keep on the screen for a while. For example, many applications implement a Find dialog box that stays visible so that the user can select Find Next several times before closing the dialog box. Figure 3 shows Word for Windows 6.0's spell-checker, a good example of a modeless dialog box. Toolbars are often implemented as modeless dialogs. Users can continue to work in the application even when a modeless dialog box is active.

Use modal dialog boxes when the application needs a specific piece of information before it can continue. Once the user supplies the necessary information or cancels the operation, the dialog box closes and the application continues. A good example of a modal dialog box is the File|Open dialog box present in most applications. In this case, the application can't work on the file until the user either tells it which file to open or cancels the process of opening a file. Once the user selects the filename or cancels the operation, there is no reason to keep the dialog box around, because it will only clutter the screen. When a modal dialog box is active, the user cannot do anything else in the application until that dialog has been closed. Use this kind of dialog box sparingly.

System modal dialog boxes are rare and should be used only when the user's response is required for some problem relating to the whole system. Chances are good you'll never use these, because when this type of modal dialog is active, the user can't do anything in *any* application until the dialog is closed. An example of this kind of dialog box is a fatal hardware error or an impending systemwide memory shortage.

Another use of the dialog box is to present informational, warning, or critical messages to users. Let's spend a couple of paragraphs on the message box, since your users will see a lot of them during their time spent with Windows!

Some message boxes simply convey information or issue a warning—"File not found," for example—and an OK button to clear the message after the user has read it. For these cases, you might display just a button that dismisses the dialog box when the user is done reading the message. Other dialog boxes want a clarification from the user before proceeding: "It's Tuesday. Fax standing order to the pizza shop?" accompanied by Yes and No buttons. Finally, some message boxes present a query with the option to answer Yes, No, or Cancel: "Overwrite existing file?"

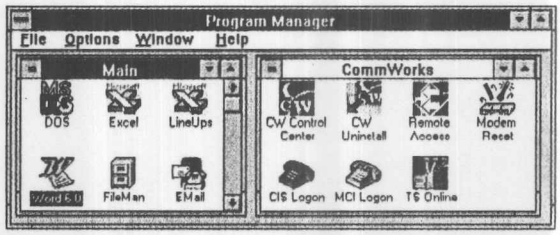

***Figure 2:* The Windows Program Manager is an MDI application that uses MDI child (document) windows for program groups. In this example, two MDI child windows—titled Main and CommWorks—are visible.**

Message boxes contain one of four standard icons. A stop sign indicates a critical error, a circled exclamation mark represents a warning message, and a circled letter *i* means an informational message. Some message boxes display a circled question mark when asking a question of the user, but since the question mark is often associated with Help, its meaning in the context of a message box is ambiguous and shouldn't be used.

DON'T BE A CONTROL FREAK Controls are those interface elements (command buttons, scroll bars, list boxes, edit boxes, and so on) that handle interaction with the user. In most cases, controls serve both to get information from the user and to display information about the current application status. For instance, scroll bars in a word processor not only let the user move around within a file but also display the user's current position within that file. Radio buttons (which are described below) both show the current selection and let the user change that selection.

Right out of the box, Windows provides a number of controls that you can use when designing an application. Although you might be tempted to use a wide variety of custom controls (including the numerous Visual Basic .VBX controls), you should carefully consider using existing controls before using a control that might be unfamiliar to users. To a new Visual Basic or PowerBuilder developer, it's really easy to lose control and try out each of the custom controls found in those environments.

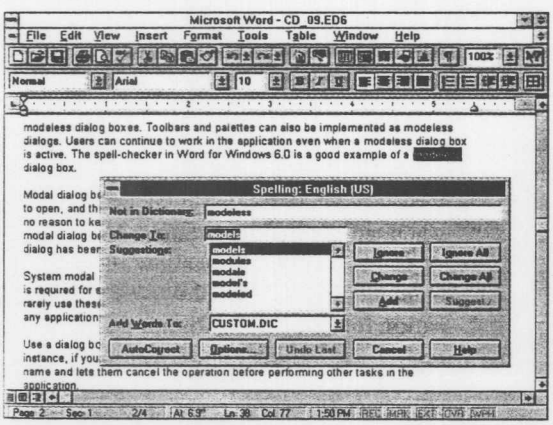

***Figure 3:* The Microsoft Word for Windows spell-checker is a good example of a modeless dialog box. You can move around the document and edit text even with the dialog box active.**

Figure 4 shows a dialog box containing some of the common Windows controls that you can use in your applications. Let's spend a few moments reviewing each of the controls and when it's appropriate to use them.

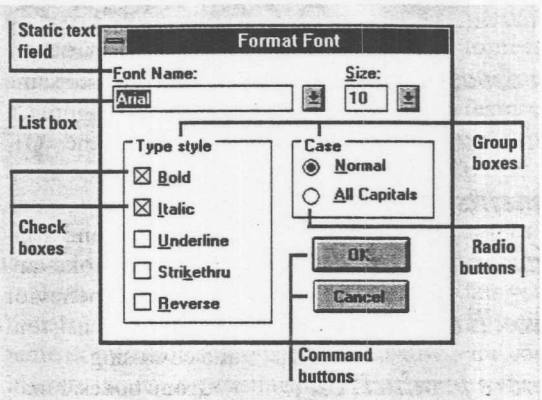

***Figure 4:* This dialog box contains most of the common Windows interface controls. Each labeled item is described in more detail in the text.**

• **Command buttons.** A command button is a rectangular button that contains a label describing a command or action that the application will carry out when the button is pressed. The button labels are usually text, but many applications also use bitmapped images on a command button. Users can either click on the command button or hold down the Alt key while pressing the underlined character that appears on the button's caption. These keyboard shortcuts are an essential part of good interface design.

Use care assigning labels for keyboard shortcuts, and ensure that no two controls in a dialog box are mapped to the same keyboard shortcut. You're heading for a train wreck if you define the keyboard shortcut of O for both the Orders command button and the OK command button. Wherever possible, use the first letter of the label as the keyboard shortcut, but if that creates a conflict with another label, choose another letter that makes it easy for the user to remember the command.

Place the standard command buttons—usually OK and Cancel—in one consistent location for all of your application's dialog boxes and the buttons for other features in a different consistent location (perhaps stacked vertically above the OK and Cancel buttons). Remember to provide keyboard shortcuts for all menu items and controls (except the OK and Cancel buttons, which should be mapped to the Enter and Esc keys, respectively). Every Windows program should be completely operable using only the keyboard, unless its function depends solely on another type of input device, such as a mouse or a puck.

• **Radio buttons.** Use radio buttons (sometimes called option buttons) as a way to present a single choice from a limited set of mutually exclusive choices. The control gets its name from the old car radio buttons of the fifties and sixties: Push one button and the others pop out. Radio buttons are represented graphically as a circle that is filled when the user chooses that option. Use radio buttons when the number of choices is limited to usually no more than four or five choices. If you need to present more than five choices, use a standard or drop-down list box.

• **Check boxes.** Choose check boxes if you want to present a list of choices to the user where they can make more than one selection from the list. Check boxes are represented by squares that display an X when the option is selected and are blank when the option is deselected. (Some applications have check boxes that use a check mark instead of an X.) As you can see in Figure 4, you can assign keyboard shortcuts to both check boxes and radio buttons—a feature your laptop-equipped mobile executives with limited mouse real estate will certainly appreciate.

• **List boxes.** A list box normally displays a column of choices. If there are too many choices to display at once, a scroll bar to the right of the list lets the user scroll through the choices. By default, list boxes let the user select only one item, although you can design a list box to permit multiple-item selection. In addition, another flavor of list box, called a combo box, lets you present

multiple items to the user but permits him to type in his own response if the correct one isn't on the list.

Because it's easy to add or subtract data elements from the list, consider using list boxes for single or multiple selections when the features or data that you present to the user is dynamic. Drop-down lists and combo boxes are an attractive option when screen real estate is limited or when the user won't need to access the list of choices very often.

- **Text boxes.** Text boxes are controls into which the user types information. Within the text box control, the user may navigate text by pressing the Left or Right Arrow key and select text by holding down the Shift key while pressing the Left or Right Arrow key. Text boxes also support navigation and selection with the mouse. Text boxes can be of the single-line or multiline variety. The Windows Notepad is just one multiline text box with a menu bar. The dialog box in Figure 1 contains four single-line text boxes.
- **Static text fields.** Be sure to label your controls with static text fields. In some cases—that is, for buttons, check boxes, and group boxes—the operating system provides labels for the controls, but it's up to you to label other controls. Keep static text strings short and align data-entry fields to make navigation easier. Be consistent when labeling controls by developing a list of reserved words for controls in order to prevent multiple meanings of the same word. For example, I have seen the OK button used in nonstandard ways, such as to close one window and invoke another. This behavior is both inconsistent and confusing.
- **Group boxes.** Finally, use group boxes to group together related controls and to provide visual consistency throughout your dialog boxes. Although group boxes are technically considered controls, the user can't manipulate them with the mouse or the keyboard. The most common use of a group box is to enclose groups of radio buttons or check boxes.

If you choose the right window and control elements for your Windows-based applications, you'll be rewarded with more productivity and fewer headaches.

HONEY, I HID THE CONTROL Consider whether you want to disable unavailable menu items and controls (leaving them visible but grayed out) or hide them. In general, graying a control is much safer. Users are often confused when menu items or controls keep appearing and disappearing. The only reason to hide GUI components is if the user will never have access to specific menu items or controls. For example, you might choose to hide certain controls in a window or dialog box based on the user's security level at log-on. Another reason to leave disabled controls in place is if you've provided context-sensitive help. That way, users can tab to the disabled control and press the F1 to get context-sensitive help in order to figure out why they can't access the control.

CONCLUSION Figure 5 summarizes the issues you should consider when using controls in dialog boxes. While Windows controls can often accommodate a large number of choices, it's not a good idea to explore that limit with your users. Note, for instance, the suggested maximum number of choices for a list box. I've found that users get frustrated after hitting PgDn or scrolling through 50 or more items. With a longer list, consider providing a method for reducing the list to create a subset from which the user can choose. A simple data-entry field where the user can enter a partial string to search on would work well, for example.

If you find that you're spending a lot of time with user-interface issues, you owe it to yourself to look into *The Windows Interface: An Application Design Guide* (Microsoft Press, ISBN: 1-55615-439-9). This interface bible is the definitive compilation of Windows UI arcana, from accelerator keys to z-order, with discussions on both the appearance and the behavior of Windows interface elements. This book is essential when setting corporate standards.

Choosing the right window and control elements for your Windows-based application forms a strong foundation for the rest of the application. If you make the right choices and don't try to get too far out onto the bleeding edge of interface technology, you will be rewarded with more productivity and fewer support headaches. Making the wrong choice, on the other hand, is the stuff GUI nightmares are made of. □

Deciding on Controls for Dialog Boxes

If you want to	Use a	But no more than...
Trigger an action	Command button	Six per dialog box
Choose among a few mutually exclusive options	Radio button	Six per group
Choose one or more of a few nonexclusive options	Check box	Ten per group
Get arbitrary text data from the user	Text box	
Choose among mutually exclusive options	List box	50 items, displayed 8-10 at a time
Choose one or more nonexclusive options	Multiple selection list box	50 items, displayed 8-10 at a time
Choose among mutually exclusive options while using a minimum of space	Drop-down list box	20 items, displayed 1 at a time
Choose among mutually exclusive options or enter a new value	Combo box	20 items, displayed 1 at a time
Group related controls, especially radio buttons and check boxes	Group box	
Label controls that don't have their own labels (list boxes, for example)	Static text field	

***Figure 5:* This table summarizes the available standard interface elements and shows recommended limits to the number of controls that you should use in a single dialog box. Too many controls and the dialog box becomes difficult to understand.**

CHRISTINE COMAFORD IS PRESIDENT OF CORPORATE COMPUTING INC., A USER-INTERFACE AND CLIENT/SERVER DEVELOPMENT CONSULTING FIRM. SHE CAN BE REACHED VIA MCI MAIL (371-9004).